Objective: Install and learn Code Composure Studio, create and build a simple program to blink an onboard LED using clocks, interrupts, and timers, open and utilize a debug session, and flash a program on the Stellaris Launchpad.

The tools and concepts covered in this example are:
- Microcontroller Basics;
- Installing CCS v5.4;
- Workspaces;
- Welcome screen / TI Resource Explorer;
- Stellaris LaunchPad;
- Hello World!;
- Debugging;
- Clocks;
- Interrupts;
- Timers.

Required Documents for this lab:
- LM4F120H5QR Data Sheet and User's Guide http://www.ti.com/lit/ds/spms351b/spms351b.pdf
- Stellaris Workshop Manual http://software-dl.ti.com/trainingTTO/trainingTTO_public_sw/GSW-Stellaris-LaunchPad/StellarisLaunchPadWorkbook.pdf
- ARM Optimizing C/C++ Compilers User Guide http://www.ti.com/lit/ug/spnu151i/spnu151i.pdf
- BoosterPack Development Guide http://www.ti.com/lit/ug/spmu288/spmu288.pdf
- Stellaris LM4F120 LaunchPad Evaluation Board User Manual http://www.ti.com/lit/ug/spmu289b/spmu289b.pdf

There are some useful documents that you should have available in C:\StellarisWare\docs and find:
- Peripheral Driver User's Guide (SW-DRL-UGxxxx.pdf)
- USB Library User's Guide (SW-USBL-UGxxxx.pdf)
- Graphics Library User's Guide (SW-GRL-UGxxxx.pdf)
- LaunchPad Board User's Guide (SW-EK-LM4F120XL-UG-xxxx.pdf)

Caution: Ensure adequate time is utilized to properly comprehend what is accomplished in each section of the lab rather than skimming or skipping. Your hard work will be rewarded.

Microcontroller Basics: Provides an overview of microcontrollers ~ (5min – 85min)
Installing CCS v5.4: Learn how to install and setup CCS v5.4 ~ (30min – 90min)
Stellaris LaunchPad: Introduces the development kit, Launchpad, and microcontroller ~ (15min)
My First Project: Basic project, blink LED program, test Stellaris, and debugging ~ (60min – 120min)
Project Two: Introduction to StellarisWare®, Initialization and GPIO ~ (60min – 80min)
Project Three: Interrupts and the Timers ~ (45min – 60min)
Finale Project: Implement clocks, interrupts, and timers together ~ (Unknown)

Microcontroller Basics
TI Stellaris LM4F120H5QR Microcontroller: A highly integrated chip that contains all the components comprising a controller: CPU, RAM, some form of ROM, I/O ports, and peripherals. The EDN (Electronics Design Network) website is a microcontroller introductory presentation and requires one hour and fifteen minutes to complete, but is not required only suggested for this class or lab. http://techonline.com/electrical-engineers/education-training/courses/4000041/Fundamentals-of-Microcontrollers

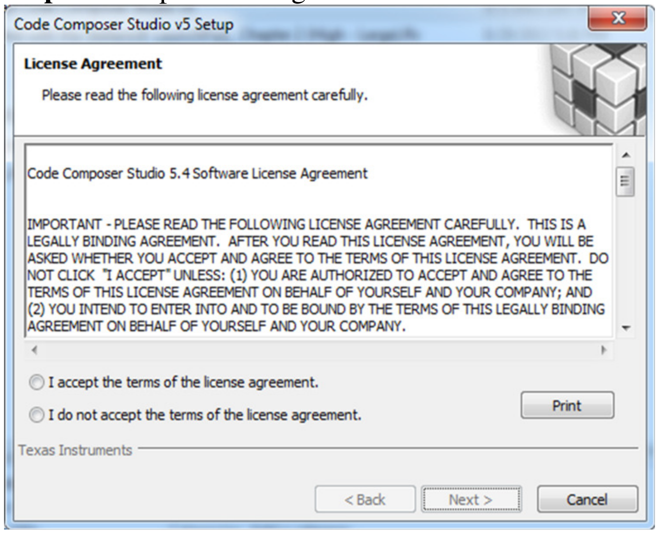
The Stellaris LM4F120H5QR is a low power 32 bit RISC MCU with 256 kb flash memory and 32kb of SRAM. It has an internal clock with frequencies up to 80 MHz, low frequency oscillator, 12 timers with PWM and CCP, 44 GPIO pins, seven exceptions and 65 interrupts, eight universal serial communication interfaces, two 12-bit analog to digital converters, and floating-point instructions are IEEE 754 compliant. The data sheet has further details and instructions on the LM4F120.

Introduction
Microcontrollers use an integrated development environment (IDE) for programming and debugging. Texas Instruments' (TI) has developed their own IDE program called Code Composure Studio™ (CCS) for it embedded processor families. Detailed information about this IDE can be found at: http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v5. There are other development tools for the LM4F120 such as Energia, however this lab will exclusively use CCS to further understanding for microcontroller development in an industry environment where Energia is for hobbyists and doesn't include debugging tools.

Installing CCS
Instructions for downloading and installing CCS are available on the TI wiki website http://processors.wiki.ti.com/index.php/Download_CCS. Please follow the steps below for this lab.

Step 1: Download the latest "Web Installer" for Windows. You may need to sign up on TI website and fill out a form taking in basic information. Double click the ccs_setup_5.x.x.xxxxx.exe (executable) file to open up an installation wizard.

Step 2: Accept license agreement terms and click Next.


Code Composer Studio v5 Setup

License Agreement

Please read the following license agreement carefully.

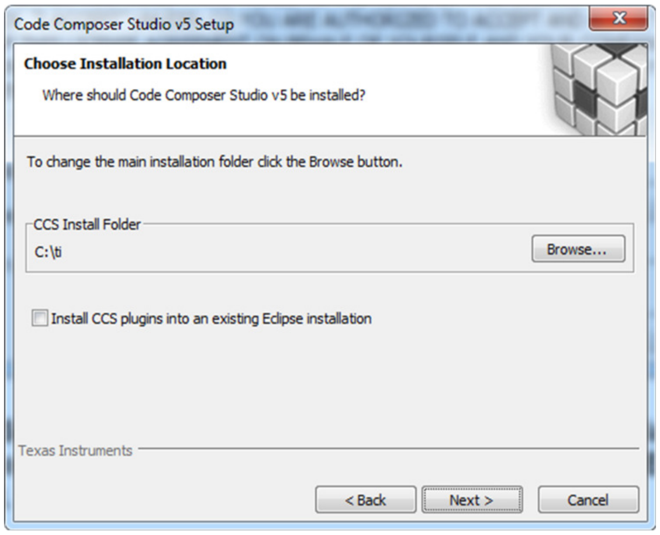
Code Composer Studio 5.4 Software License Agreement

IMPORTANT - PLEASE READ THE FOLLOWING LICENSE AGREEMENT CAREFULLY. THIS IS A LEGALLY BINDING AGREEMENT. AFTER YOU READ THIS LICENSE AGREEMENT, YOU WILL BE ASKED WHETHER YOU ACCEPT AND AGREE TO THE TERMS OF THIS LICENSE AGREEMENT. DO NOT CLICK "I ACCEPT" UNLESS: (1) YOU ARE AUTHORIZED TO ACCEPT AND AGREE TO THE TERMS OF THIS LICENSE AGREEMENT ON BEHALF OF YOURSELF AND YOUR COMPANY; AND (2) YOU INTEND TO ENTER INTO AND TO BE BOUND BY THE TERMS OF THIS LEGALLY BINDING AGREEMENT ON BEHALF OF YOURSELF AND YOUR COMPANY.

I accept the terms of the license agreement.

I do not accept the terms of the license agreement.

Step 3: Select "CCS Install Folder" by browsing appropriate folder. Keep the check box labeled "Install CCS plugins into an existing Eclipse installation" un-checked. Click Next.


Code Composer Studio v5 Setup

Choose Installation Location

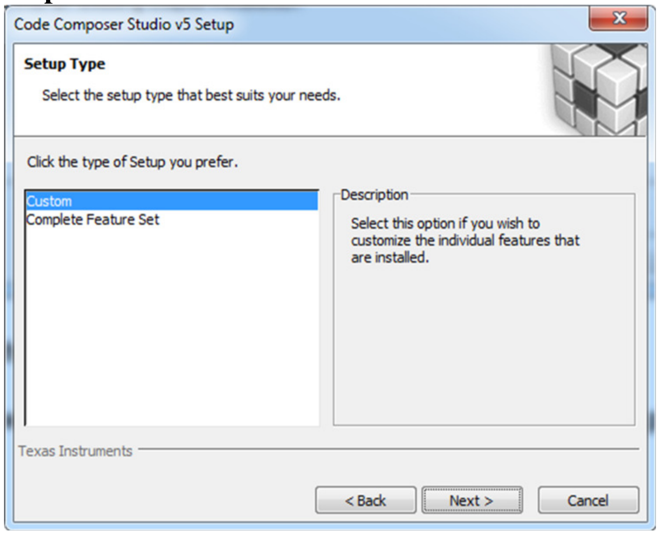
Where should Code Composer Studio v5 be installed?

To change the main installation folder click the Browse button.

CCS Install Folder: C:\ti

☐ Install CCS plugins into an existing Eclipse installation

Step 4: Select "Custom." Click Next.


Code Composer Studio v5 Setup

Setup Type

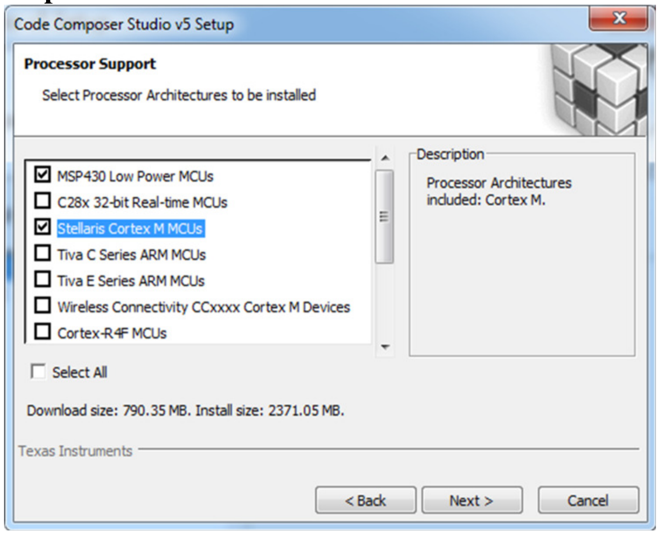
Select the setup type that best suits your needs.

Click the type of Setup you prefer.

- Custom
- Complete Feature Set

Description: Select this option if you wish to customize the individual features that are installed.

Step 5: Check the boxes selected below. Click Next.


Code Composer Studio v5 Setup

Processor Support

Select Processor Architectures to be installed

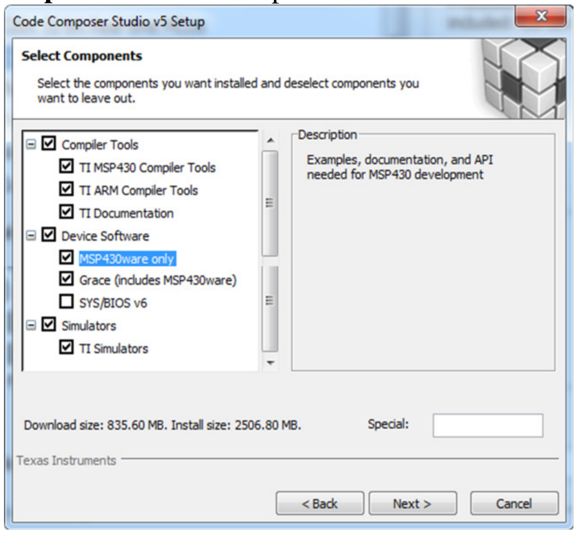
- ☑ MSP430 Low Power MCUs
- ☐ C28x 32-bit Real-time MCUs
- ☑ Stellaris Cortex M MCUs
- ☐ Tiva C Series ARM MCUs
- ☐ Tiva E Series ARM MCUs
- ☐ Wireless Connectivity CCxxxx Cortex M Devices
- ☐ Cortex-R4F MCUs

Description: Processor Architectures included: Cortex M.

☐ Select All

Download size: 790.35 MB. Install size: 2371.05 MB.

Step 6: Select the components selected below. Click Next.


Code Composer Studio v5 Setup

Select Components

Select the components you want installed and deselect components you want to leave out.

- [✓] Compiler Tools
  - [✓] TI MSP430 Compiler Tools
  - [✓] TI ARM Compiler Tools
  - [✓] TI Documentation
- [✓] Device Software
  - [✓] MSP430ware only
  - [✓] Grace (includes MSP430ware)
  - [ ] SYS/BIOS v6
- [✓] Simulators
  - [✓] TI Simulators

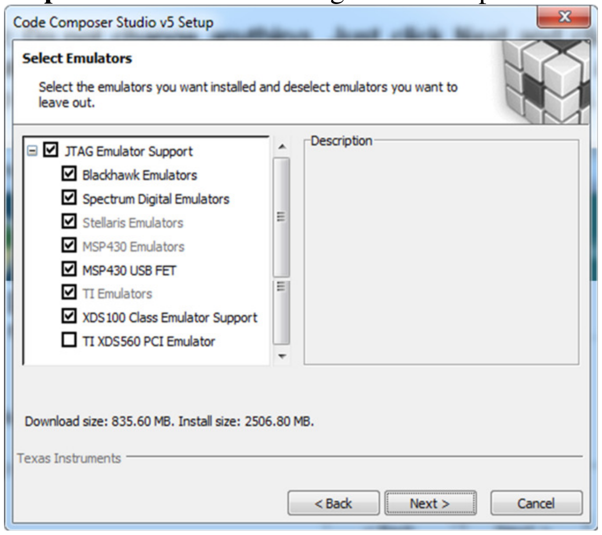
Description: Examples, documentation, and API needed for MSP430 development

Download size: 835.60 MB. Install size: 2506.80 MB.

Special:

Texas Instruments

Step 7: Ensure the following emulator options are selected. Click Next.


Code Composer Studio v5 Setup

Select Emulators

Select the emulators you want installed and deselect emulators you want to leave out.

- [✓] JTAG Emulator Support
  - [✓] Blackhawk Emulators
  - [✓] Spectrum Digital Emulators
  - [✓] Stellaris Emulators
  - [✓] MSP430 Emulators
  - [✓] MSP430 USB FET
  - [✓] TI Emulators
  - [✓] XDS100 Class Emulator Support
  - [ ] TI XDS560 PCI Emulator

Description

Download size: 835.60 MB. Install size: 2506.80 MB.

Texas Instruments

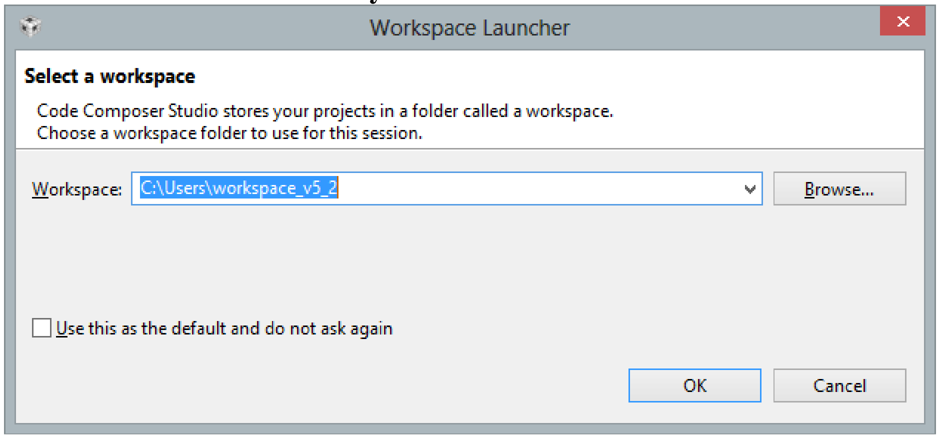
Step 8: This is the summary screen for all options selected. Just click Next to begin Installation. It may take a several minutes to install. Allow access to Java Platform if a pop-up appears during installation (see picture below). There may be prompts for firewall access for the different components to talk to each other and prompts to install different components.

Step 9: Click Finish to launch Code Composer Studio v5.

Workspace

Executing CCS requires setting a workspace, shown in Fig. 1.1, and is the directory that contains all components (i.e., projects, links to projects, possibly source code, etc.) used in the development. The default workspace is created under the C:\Users\<user> for Windows 7/8 or C:\Documents and Settings\<user> directory for Windows XP, however the user has the liberty to alter the workspace location. Utilizing a single directory for all projects requires checking the option “Use this as the default and do not ask again.” The workspace maybe altered later inside CCS. Maintaining multiple workspaces is possible, however there a few limitations. Only one active workspace within each CCS instance and sharing the same workspace across multiple running CCS instances is not permitted. It is not encouraged or recommended to share workspaces amongst users.

Note: Please save all work to your H drive or it will be erased.


Workspace Launcher

Select a workspace

Code Composer Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\workspace_v5_2

Fig. 1.1. The CCS launch tool for setting the workspace.

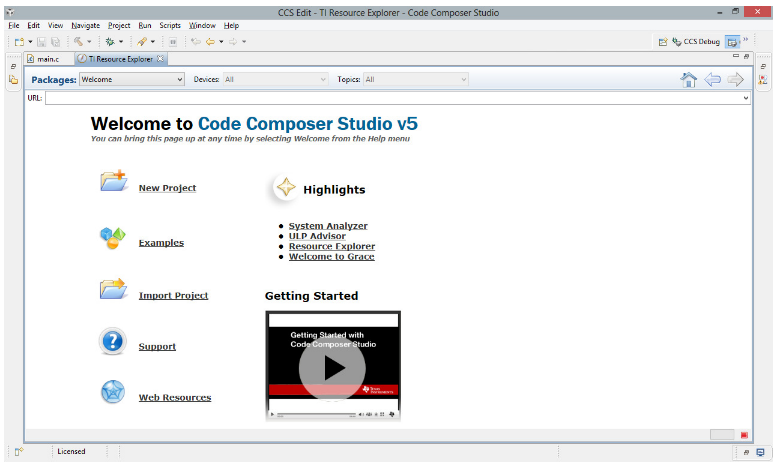
CCS License options

There are a number of different CCS license options and their details are shown online at http://processors.wiki.ti.com/index.php/Licensing_-_CCS. CCS requires an active license on first execution. There are two different licenses to select from for CCS: CODE SIZE LIMITED (MSP430) and the FREE LICENSE for use with the onboard emulators on Stellaris development kits. This lab manual will apply the **FREE LICENSE** for Stellaris development. The free license option allows the usage of CCSv5.4 with Stellaris microcontrollers at no charge, no need to for an active internet connection, and may be used from the beginning since it is activated directly on the PC;

1. Select the option **free license** for Stellaris launchpad and click Finish.
2. Once the license install has completed, restart CCS and the bottom display bar should read "Licensed."

Welcome Window

After the license process and CCS is restarted, CCS will display the self-explanatory welcome window as shown in Fig. 1.2. and is ready to use. The welcome package shows several links and a short "Getting Started" clip. For future reference to open the welcome window again click on the "view" menu option and select "TI Resource Explorer" and then under packages select "Welcome."



Welcome to Code Composer Studio v5

You can bring this page up at any time by selecting Welcome from the Help menu

Fig. 1.2. CCS welcome window seen at first time launch.

Install the following software components for use with this lab, unless already installed.

Download and Install StellarisWare

Download and install the latest full version of StellarisWare from: http://www.ti.com/tool/sw-lm3s.

Download and Install LM Flash Programmer

Download, unzip and install the latest LM Flash Programmer (LMFLASHPROGRAMMER) from http://www.ti.com/tool/lmflashprogrammer.

Download and Install ICDI Drivers

Download the latest version of the in-circuit debug interface drivers from http://www.ti.com/tool/stellaris_icdi_drivers. Unzip the file and place the stellaris_icdi_drivers folder in C:\StellarisWare.

Terminal Program

Download PuTTY: http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe

Note: if controlSUITE, MSP430Ware, etc... is previously installed in your system, their examples will be displayed at the link Examples.

The MSP430Ware package contains useful information for the inspiring young developer:

- Collection of code examples, datasheets and other design resources for ALL MSP430 devices;
- Provides a GUI for navigating all existing MSP430 design resources;
- Includes a high-level application programming interface (API) called Driver Library, which makes it really easy to talk to MSP430 hardware.

Additional information available online at http://www.ti.com/tool/msp430ware.

Enter Launchpad

The following subsection introduces CSS project development, Stellaris programming insight and edification in conjunction with a “Hello World” program, and introducing project debugging for the Stellaris experimenter board. The Stellaris launch pad is a low-cost experiment board providing a complete development environment featuring integrated USB-based emulation and all the hardware and software necessary to develop applications for the Stellaris. Fig. 1.3 introduces the Launchpad development board and includes annotations to increase familiarity with microcontroller development. The Stellaris launch pad kit containing the following items: launch pad emulator socket board (EK-

LM4F120XL), micro USB-B cable, installed LM4F120H5QR, two female headers attached on the underside, one 32.768 kHz micro crystal pre-soldered, one static free bag for Launchpad, and quick start guide. Please make sure the kit issued has all the required components before continuing the lab. The Stellaris launch pad may be purchased inexpensively from TI for your own continual edification and exploration.

Note: TI has replaced and rebranded the Stellaris with TIVA™ C-Series TM4C123G LaunchPad. The TIVA TM4C1233H6PM and Stellaris LM4F120H5QR are identical. (As of September 2013)

Additional details about the hardware development tool can be found in:

- Main page: www.ti.com/launchpad
- Stellaris LaunchPad: www.ti.com/stellaris-launchpad
- EK-LM4F120XL product page: http://www.ti.com/tool/EK-LM4F120XL
- BoosterPack webpage: www.ti.com/boosterpack
- LaunchPad WiKi: www.ti.com/launchpadwiki
- LM4F120H5QR folder: http://www.ti.com/product/lm4f120h5qr

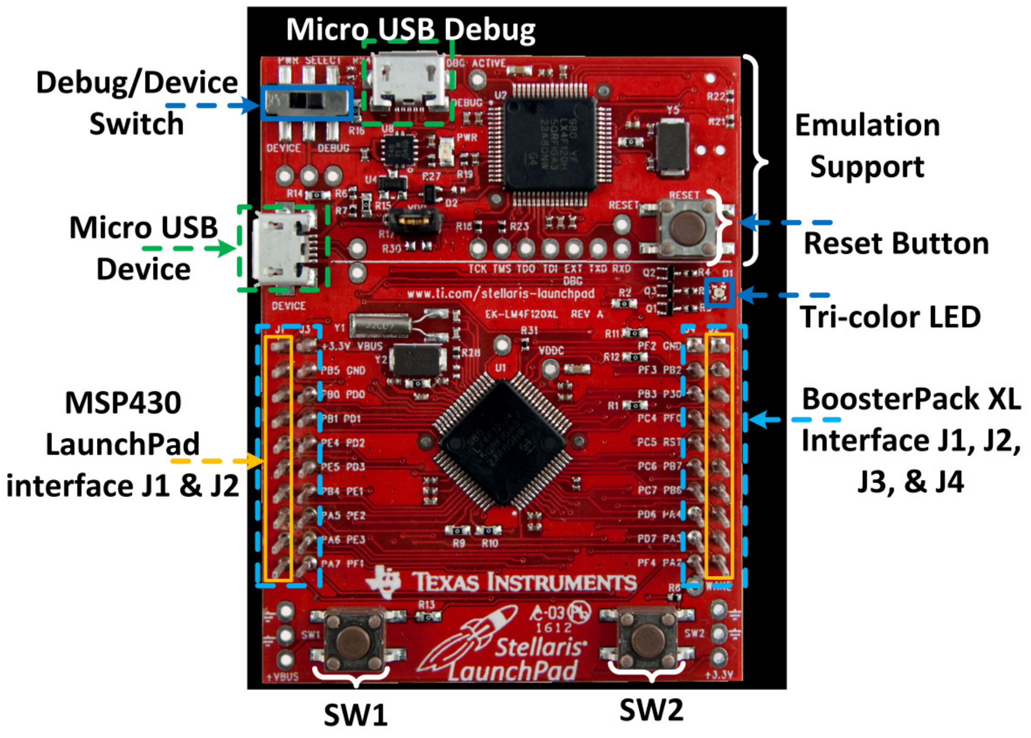


Fig. 1.3. Stellaris Launchpad board annotated for identification ease.

Connecting the board and installing the drivers

The LM4F120 LaunchPad Board ICDI USB port (marked DEBUG and shown above) is a composite USB port and consists of three connections:
Stellaris ICDI JTAG/SWD Interface - debugger connection
Stellaris ICDI DFU Device - firmware update connection
Stellaris Virtual Serial Port - a serial data connection

Testing New Stellaris Launchpad

The LM4F120H5QR arrives pre-programmed with an LED toggle sequence program. Using the micro USB cable, connect the USB emulation connector (marked DEBUG) to an available USB port. A PC's

USB port will output up to 500 mA for each device attached and sufficient for the LaunchPad. Connecting the board with a USB hub requires an external power source. The Launchpad drivers should install automatically. Once this completes, the board will be lit with two LEDs. One green LED off by itself confirms the board is powered; the other LED is a RGB LED experimental component that is cycling through its color spectrum with the pre-programed experiment. The two pushbuttons at the bottom of your board are marked SW1 and SW2. Press or press and hold SW1to advance the light spectrum toward the red portion of the color spectrum. Press or press and hold SW2 to advance the light toward the violet portion of the color spectrum. When no switch is pressed for five seconds, the software returns to automatically changing the color display. Press and hold both SW1 and SW2 for three seconds to enter hibernate mode. While in this mode, the last color will blink on the LEDs for ½ second every 3 seconds. Between blinks, the device enters the VDD3ON hibernate mode with the realtime-clock (RTC) running. Pressing SW2 at any time will wake the device and return to automatically changing the color display. This verifies the pre-loaded Stellaris MCU is operating correctly.

What Happens at Power On

A power-on reset (POR) begins when the device is powered on or a low signal occurs at the reset pin while powered when configured for reset mode. The Stellaris must go through a series of steps before the software launches.

1. At power-up, the brown-out circuitry holds the device in reset until VDD is above a hysteresis point. The brown-out reset circuit detects low supply voltages such as when a supply voltage is applied to or removed from the VDD terminal. The brown-out reset circuit resets the device by triggering a POR signal when power is applied or removed.
2. Internal reset is released
3. The core loads from memory the initial stack pointer, the initial program counter, and the first instruction designated by the program counter, and then begins execution
4. Pins configured back to default assignments
5. Clocks are configured
6. Peripheral modules and registers are initialized
7. Status registers (SR) are reset
8. Watchdog timers power up in watchdog mode.
9. Initial Program counter (PC) is loaded and begins execution

Software Initialization follows system reset

A software reset begins with POR signal, the watchdog timer has expired when enabled in supervision mode, or a flash memory access control registers security key violation has occurred.

1. Initialize the stack pointer (SP)
2. Reconfigure clocks (if desired)
3. Initialize the watchdog timer(s) to the requirements of the application, usually off for debugging
4. Configure peripheral modules

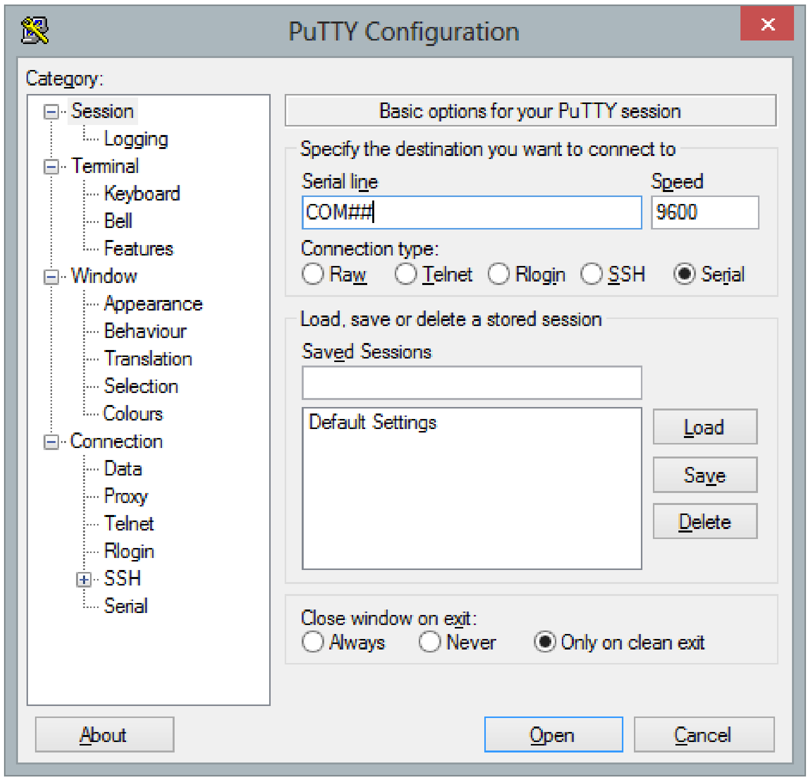
Power-On and Brown-Out Reset Control (PBORCTL) Register

Stellaris devices have a reset circuit to detect a power source disturbance. The PBORCTL is an enhanced POR system utilizing a hysteresis circuit to allow the device to stay in reset mode until the voltage (VDD) is higher than the upper threshold (V_{BORnTH}). When VDD is higher than V_{BORnTH}, the BOR takes 2 msec to deactivate and permit CPU program execution. If VDD decreases below the lower threshold V_{BORnTH}, caused either by power source interruption or battery discharge, the BOR circuit will generate a reset signal and suspends processor operation until the VDD rises above V_{BORnTH}. For additional information and graphs, please see the Stellaris User's Guide section 5.2.2.4, pg 233, and 22.6.

Establishing a UART Connection

UART is connected as a virtual serial port through the emulator USB connection. The following steps will demonstrate opening a connection to the LaunchPad using PuTTY in Windows 7. Identify the COM port number of the Stellaris Virtual Serial Port in Device Manager by right-clicking the “Computer” icon located on the desktop and selecting “Manage.” Then in the left pane select “Device manager.” Expand the “Ports (COM & LPT)” component and the see the entry titled “Stellaris Virtual Serial Port (COM##).

Next, double-click on PuTTY located under start menu. Make the settings shown in Fig. 1.4 and then click Open. Change the ## to your COM port number. Set the COM port speed at 115200. When the terminal window opens, press Enter once and the LaunchPad board will respond with a > indicating that communication is open. Type “help” and a list of commands will be displayed. Please try them. When finished, close the PuTTY terminal window.



Serial line: COM##
Speed: 9600
Connection type: Raw Telnet Rlogin SSH [✓ Serial]
Close window on exit: Always Never [✓ Only on clean exit]

Fig. 1.4 PuTTY settings for initialing UART terminal communications with the Stellaris.

<u>My First Project</u>

Creating a Project

1. Open CCS and select the active workspace.
2. Click on file on the menu and under new select CCS Project, see Fig. 1.5.
3. This opens the CCS project wizard shown in Fig. 1.6. Name your project, for example Blink_LED.
4. Output type is executable. The executable setting builds a complete executable program on the MCU. The other option is static library, which is a collection of functions to be used by other projects.
5. Choose the location, default is the workspace chosen when CCS was opened.
6. Under the device section choose ARM under family.

7. Under variant type 120
8. Then examine the MCU on the Launchpad and choose the correct model number
   Note: if the device variant chosen is specific, the project wizard will enable the option Connection. This allows selecting which emulator, board, or simulator will debug the project.
9. Connection is for the Stellaris In-Circuit Debug Interface.
10. Skip advanced settings and select "Empty Project" under project templates and examples. The Project Templates and examples section contains the standard C and assembly projects, as well as several templates to create projects that use DSP/BIOS, SYSBIOS, IPC, etc. or some pre-configured basic examples (One of them very similar to which will be developed in this lab). Additional reading about project templates can be found at: http://processors.wiki.ti.com/index.php/Project_Templates_in_CCS
11. Click finish!

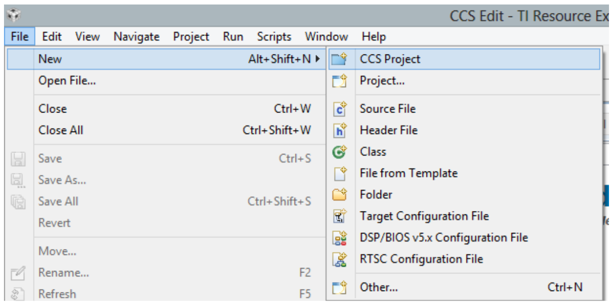
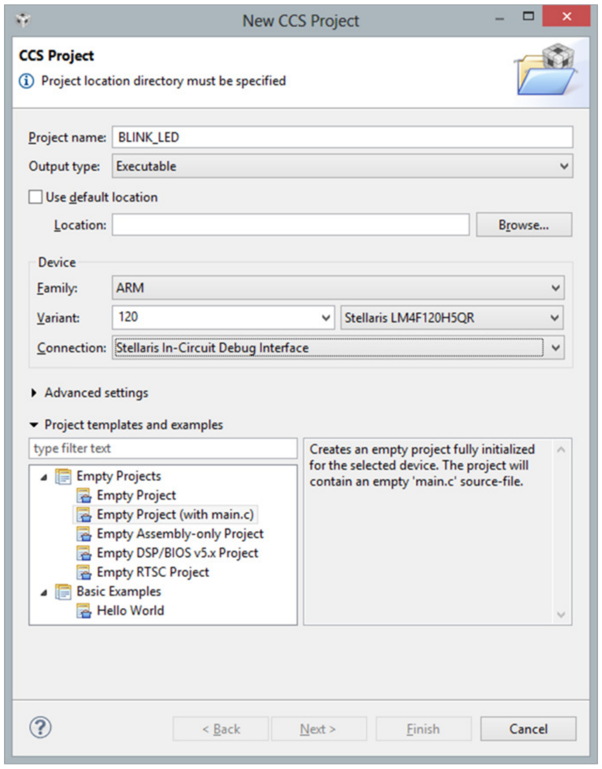


Fig. 1.5. Creating a new project in CCS.



Project location directory must be specified

Project name: BLINK_LED
Output type: Executable
Use default location
Location:
Family: ARM
Variant: 120 Stellaris LM4F120H5QR
Connection: Stellaris In-Circuit Debug Interface

- Empty Projects
  - Empty Project
  - Empty Project (with main.c)
  - Empty Assembly-only Project
  - Empty DSP/BIOS v5.x Project
  - Empty RTSC Project
- Basic Examples
  - Hello World

Creates an empty project fully initialized for the selected device. The project will contain an empty 'main.c' source-file.

Fig. 1.6. CCS project creation wizard.

- Projects map to file system directories;
- Files can be added or linked to the project;
  - Adding file to project: Copies the file into your project folder;
  - Linking file to project: Makes a reference to the file in your project but the file stays in its original location.
- Projects are either open (usual situation) or closed (still defined to the workspace, but it cannot be modified by the Workbench);
- Projects that are not part of the workspace must be imported into the active workspace before they can be opened (http://processors.wiki.ti.com/index.php/Importing_Projects_into_CCS);

Adding the Source File

To view your project, close the TI Resource Explorer and select Project Explorer under the view menu. The created project will be displayed in the Project Explorer view without a file <main.c>. To create the missing <main.c> file for the project, right-click on the project name in the Project Explorer view and select "New" shown in Fig. 1.7 and select Source File. Fig. 1.7 shows the new source file wizard. Fig. 1.9 shows the final project structure for this lab.

To add existing source files to the project, go to Project from the menu and then select "Add Files." A window will open to allow selecting the source files. Once the file is selected, a window will be shown allowing either Add (copy) or Link (create a shortcut) to the file. The option Create link locations relative to: is used to create portable projects. Always plan to use it when linking files to the project. Further details can be found at: http://processors.wiki.ti.com/index.php/Portable_Projects.

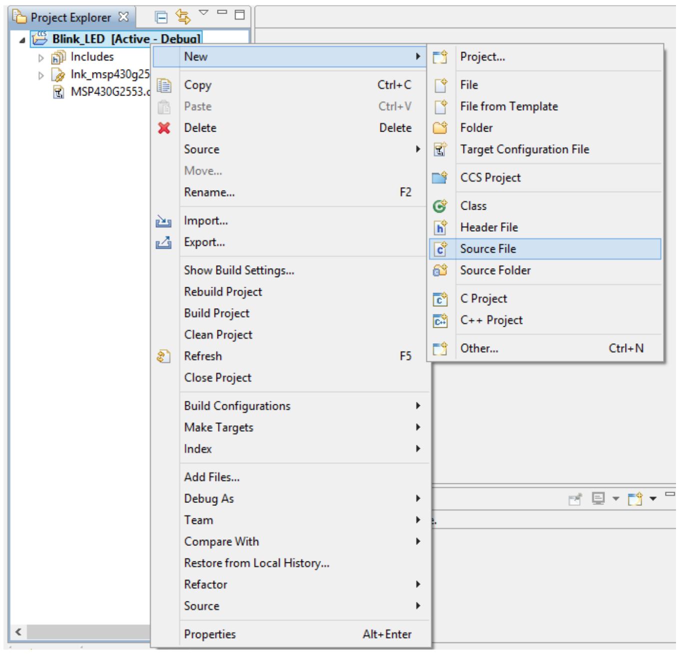
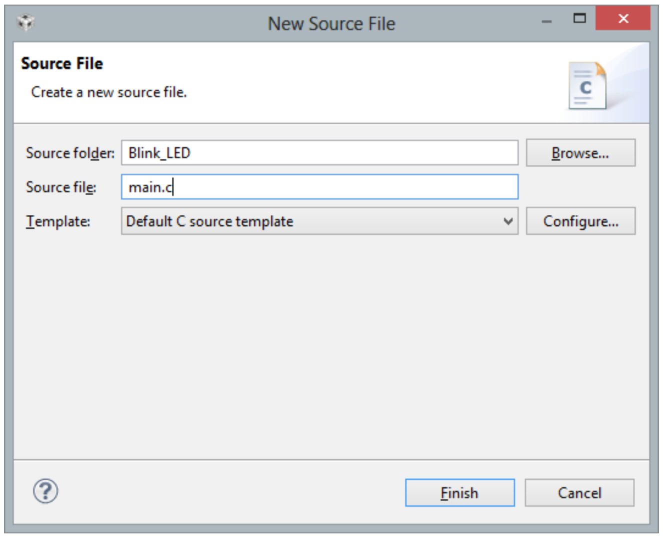


Fig. 1.7. Creating missing source file for new project.


New Source File

Source File

Create a new source file.

Source folder: Blink_LED [Browse...]

Source file: main.c

Template: Default C source template [Configure...]

Fig. 1.8. Wizard for creating a new source file.

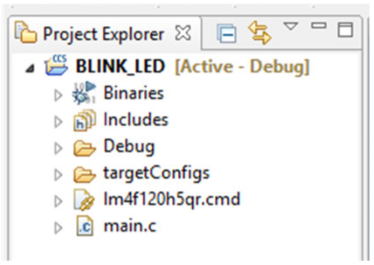


Fig. 1.9. Final project structure.

Writing the Program
This program is the "Hello World!" example for microcontrollers.
To write a comment in C, use // for one line or /* text */ for multiline comments.
At the top of the main.c file the following code adjusted as needed.

```
/************************************************************************************
Stellaris Blink Tri-color LED / Start Stop Blinking with delays - Software Toggle

 Description; Toggle inside of a software loop to start/stop blink

 MCU Model Number: LM4F120H5QR
 -----------------
 Pin Assignments
 |PF1|-->Red LED
 |PF2|-->Blue LED
 |PF3|-->Green LED

 Name
 ECEN 403 Senior Design
 Texas A&M University
 Date: September 2013
 Built with Code Composer Studio v5.4
 Version 1

 Revision information listed below:
 V1: No revisions yet
 ************************************************************************************/
```

This code above describes what this program will do, a brief layout of what pins

```
/*Compiler directives are commonly placed in the beginning of a source code and are
 *preceded by a # sign. It expands the C programming environment by providing a
 *mini-language to communicate with the compiler. */

#include <inc/hw_types.h>
//Defines common types and macros such as tBoolean and HWREG(x).

#include <inc/hw_memmap.h>
/*Macros defining the memory map of the Stellaris device. This includes
defines such as peripheral base address locations such as GPIO_PORTF_BASE.*/

#include <driverlib/sysctl.h>
/*Defines and macros for System Control API of DriverLib. This includes
API functions such as SysCtlClockSet and SysCtlClockGet.*/

#include <driverlib/gpio.h>
/*Defines and macros for GPIO API of DriverLib. This includes API functions
such as GPIOPinTypePWM and GPIOPinWrite.*/

int main(void) //Main function CCS will add { } automatically when int main() is typed.
{
      int PinData = 2; /*Creates an integer variable called PinData and initializes it to 2.
```

This will be used to cycle through the three LEDs, lighting them one at a time.*/

/*Before calling any peripheral specific driverLib function, enable the clock
for that peripheral. If not enabled, it will result in a Fault ISR (address fault).
This is a common mistake for new Stellaris users.*/

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
```

/*Using the launchpad schematic, locate the GPIO layout. You will see grouping labeled PA#, PB#, PC#, PD#, PE#, and PF#. PF1, PF2, and PF3 connect the red, blue, and green LED lights on the tri-color LED.
Now open the sysctl.h file and look for "#define SYSCTL_PERIPH_GPIO." There are several entries A to S.
Since the tri-color LED is connected to GPIOF this is the peripheral to enable.*/

```
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
```

/*This statement configures the three GPIO pins connected to the LEDs as outputs. Open up the gpio.h header file and locate the hardware calls for the pins required for the tri-color LED. Next locate "extern void GPIOPinTypeGPIOOutput(unsigned long ulPort, unsigned char ucPins);" this function defines selects the direction of the corresponding I/O port and pins. To define the port open hw_memmap.h header file containing Stellaris memory map information. There are two port access modes of GPIO ports. GPIO ports accessed through the high speed port toggle every clock cycle vs. once every two cycles for peripheral ports. In power sensitive applications, the peripheral port works better than the high speed port. For this exercise use GPIO_PORTF_BASE. For more information, see the Stellaris datasheet section 10.4. Using the bitwise OR (|) for each pin to be enabled turns off that pin. For more on the GPIO direction register see the Stellaris datasheet pg. 635.*/

```
        while(1) //Means the loop will repeat forever
        {
                //Turn on the LED
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, PinData);
```

/*Now go to the gpio.h header file and locate "extern void GPIOPinWrite(unsigned long ulPort, unsigned char ucPins, unsigned char ucVal);" this function call will turn on the tri-color LED. Declare which port and pins will be written. Make sure to read and understand how the GPIOPinWrite function is used in the Datasheet. The third data argument is not simply a 1 or 0, but represents the entire 8-bit data port. From Stellaris datasheet pg. 634, "The GPIODATA register is the data register. In software control mode, values written in the GPIODATA register are transferred onto the GPIO port pins if the respective pins have been configured as outputs through the GPIO Direction (GPIODIR) register (see page 635)."

In this example, you are writing the value in the PinData variable to all three GPIO pins connected to the LED. Only those three pins will be written based on the bit mask specified.

What are the hexadecimal values for the GPIO pins used?
Hint: There are three values and they are located in the gpio.h header file.

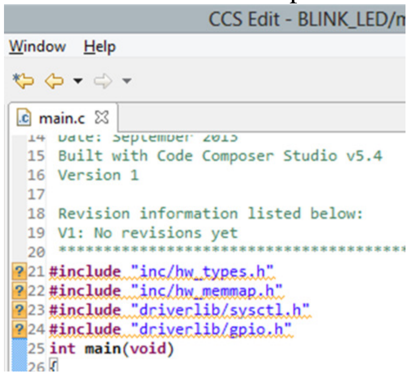
How many digits are located after the 0x?
How many writeable positions are available in the GPIODATA register?
Does the GPIODATA register use decimal, hexadecimal, octal, or binary digits?

Go to http://www.binaryhexconverter.com/decimal-to-binary-converter and convert the GPIO pin decimal values into binary and then look at how those bits would fit in the GPIODATA register

```
and which pins the value would place high.  Now is a good time to look at the Stellaris datasheet
and look through the GPIO chapter to understand the way the GPIO data register is designed.*/

        SysCtlDelay(2000000);
        /*Loop timer provided in StellarisWare. The count parameter is the
         * loop count, not the actual delay in clock cycles.*/
        // Cycle through Red, Green and Blue LEDs

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, ???);
    /*What value is used to "turn off" the pins? */

        SysCtlDelay(2000000);

        if (PinData == ???) {PinData = ???;} else {PinData = PinData*2;}
        /*This statement reads "if PinData equals eight then set PinData equal to two otherwise
        take the current PinData value, multiply it by two and redefine the value of PinData based
        on the product. */
        What is the minimum number PinData should hold and why?
        What is the maximum number PinData should cycle through and why?
        Hint: Examine the GPIO pin values
    }
}
```

TIP: If the code indentation doesn't look right, press Ctrl-A (on your keyboard) to select all the code. Then right-click on it and select "Source" then select "Correct Indentation."

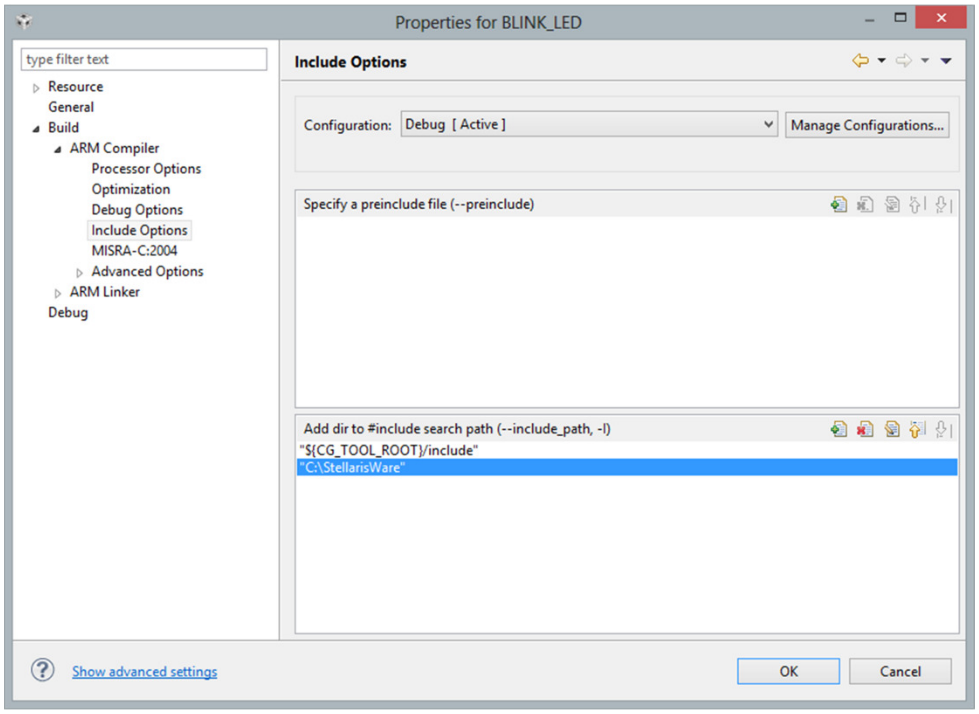
Note the question marks appearing to the left of the include statements shown in Fig.1.10. These indicate CCS does not know the path to these resources. To correct, follow the next two steps.



```
Date: September 2013
Built with Code Composer Studio v5.4
Version 1

Revision information listed below:
V1: No revisions yet
****************************************
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
int main(void)
{
```

Fig. 1.10

Step 1: Right-click on BLINK_LED in the Project Explorer pane and select "Properties" (the leftmost pane in CCS). Click "Include Options" under "ARM Compiler." In the #include search path pane, click the "Add" button and add the following include search path.
"C:\StellarisWare"
This path allows the compiler to correctly find the driverlib and inc folders. Note that if the path is not correct, this link will not work.


Properties for BLINK_LED

type filter text
- Resource
- General
- Build
  - ARM Compiler
    - Processor Options
    - Optimization
    - Debug Options
    - Include Options
    - MISRA-C:2004
    - Advanced Options
  - ARM Linker
- Debug

Include Options

Configuration: Debug [Active] Manage Configurations...

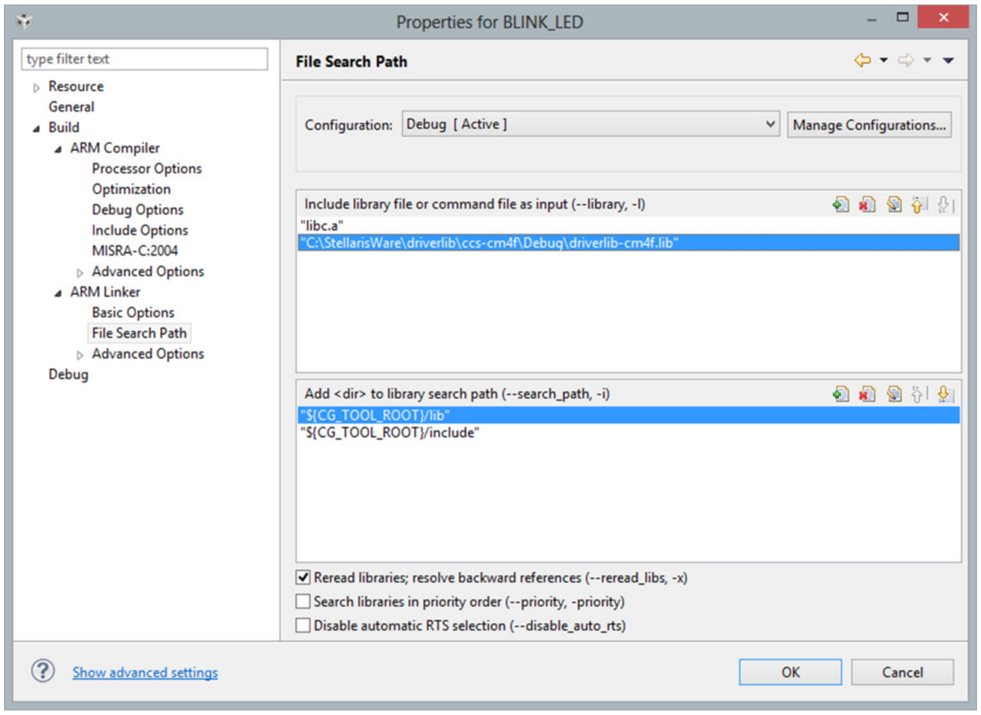
Specify a preinclude file (--preinclude)

Add dir to #include search path (--include_path, -I)
"${CG_TOOL_ROOT}/include"
"C:\StellarisWare"

Show advanced settings OK Cancel

Fig. 1.11. ARM Compiler include options for locating driverlib directory.

Step 2: Expand “ARM Linker” and click “File Search Path.” Add the following include library file to the top window:
"C:\StellarisWare\driverlib\ccs-cm4f\Debug\driverlib-cm4f.lib"
This step allows the linker to correctly find the lib file. Note that if the path is not correct, this link will not work. Click OK to save your changes.


Properties for BLINK_LED

type filter text
- Resource
- General
- Build
  - ARM Compiler
    - Processor Options
    - Optimization
    - Debug Options
    - Include Options
    - MISRA-C:2004
    - Advanced Options
  - ARM Linker
    - Basic Options
    - File Search Path
    - Advanced Options
- Debug

File Search Path

Configuration: Debug [Active] Manage Configurations...

Include library file or command file as input (--library, -l)
"libc.a"
"C:\StellarisWare\driverlib\ccs-cm4f\Debug\driverlib-cm4f.lib"

Add <dir> to library search path (--search_path, -i)
"${CG_TOOL_ROOT}/lib"
"${CG_TOOL_ROOT}/include"

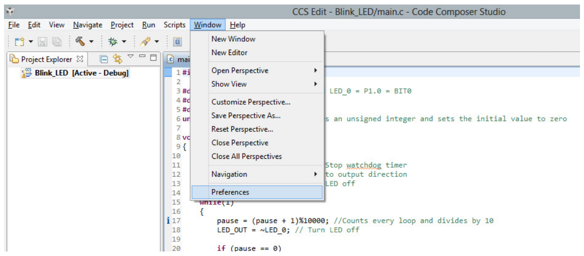
☑ Reread libraries; resolve backward references (--reread_libs, -x)
☐ Search libraries in priority order (--priority, -priority)
☐ Disable automatic RTS selection (--disable_auto_rts)

Show advanced settings OK Cancel

Fig. 1.12. ARM Linker search path options for Stellaris library file.

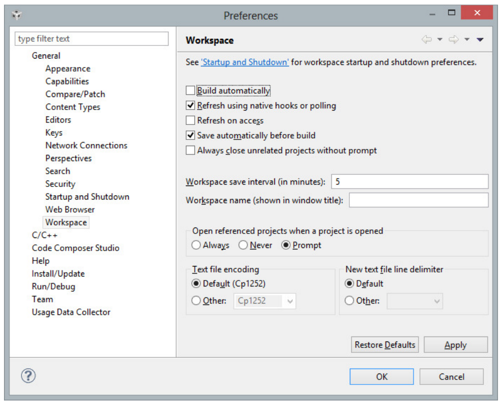
Debugging

CCS can automatically save modified source files, build the program, open the debug view, connect and flash the MCU, and then runs the program. To configure automatic options for saving and building select the "Window" dropdown menu and select "Preferences", see Fig. 1.13. The preferences for CCS open and select "Workspace" from the left menu column under the "General" options menu. This will open the setting related to automatic saving and building shown in Fig. 1.14. There are many other options and settings available to customize CCS to the user's liking and programming environment.



```
LED_0 = P1.0 = BIT0
s an unsigned integer and sets the initial value to zero
Stop watchdog timer
to output direction
LED off
while(1)
{
    pause = (pause + 1)%10000; //Counts every loop and divides by 10
    LED_OUT = ~LED_0; // Turn LED off

    if (pause == 0)
```

Fig. 1.13. Selecting the CCS preferences option.



Workspace

See 'Startup and Shutdown' for workspace startup and shutdown preferences.

- ☐ Build automatically
- ☑ Refresh using native hooks or polling
- ☐ Refresh on access
- ☑ Save automatically before build
- ☐ Always close unrelated projects without prompt

Workspace save interval (in minutes): 5

Workspace name (shown in window title):

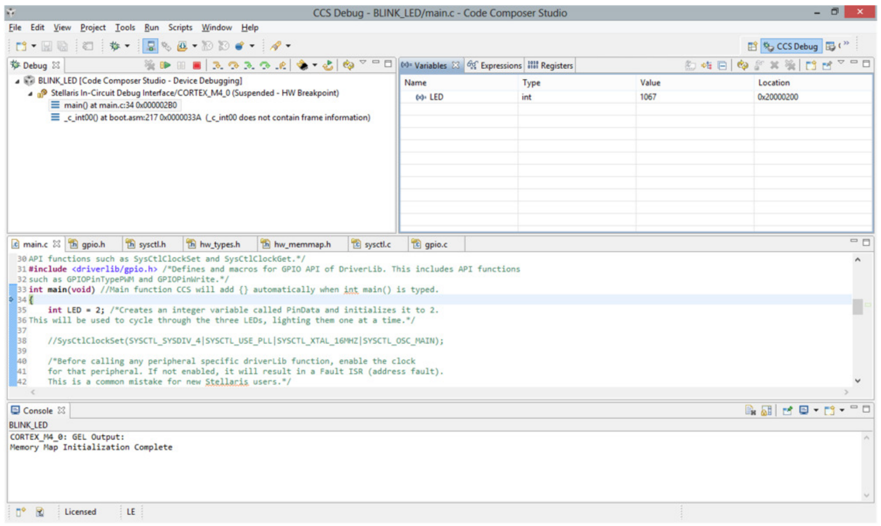
Open referenced projects when a project is opened: Always Never [✓ Prompt]

Text file encoding: [✓ Default (Cp1252)] Other: Cp1252

New text file line delimiter: [✓ Default] Other:

Fig. 1.14. CCS preferences.

To begin debugging the main.c file, select the "Debug" button located in the shortcut tool bar in CCS. If the main.c file is not saved, a pop-up box will prompt saving the project before building. The CCS compiler translates written code into machine code for the MCU to execute. The build process includes preprocessing, compiling, and linking. The build process generates the final program executable to be flashed onto the MCU. Any errors or warnings are posted in the lower right-hand window under the tab labeled "Problems". Notes from the building process are located in the console section of CCS. Once the build process is complete, the CCS debug view opens as shown in Fig. 1.15. CCS automatically resets and flashes the MCU, but will not start the program. Building the project separately without using the debugging tools is possible by right clicking on the project "BLINK_LED" under project explorer shown in Fig. 1.16 and selecting "Build Project".



```
BLINK_LED [Code Composer Studio - Device Debugging]
  Stellaris In-Circuit Debug Interface/CORTEX_M4_0 (Suspended - HW Breakpoint)
    main() at main.c:34 0x000002B0
    _c_int00() at boot.asm:217 0x0000033A (_c_int00 does not contain frame information)
```

Name	Type	Value	Location
LED	int	1067	0x20000200

```
API functions such as SysCtlClockSet and SysCtlClockGet.*/
#include <driverlib/gpio.h> /*Defines and macros for GPIO API of DriverLib. This includes API functions
such as GPIOPinTypePWM and GPIOPinWrite.*/
int main(void) //Main function CCS will add {} automatically when int main() is typed.
{
    int LED = 2; /*Creates an integer variable called PinData and initializes it to 2.
This will be used to cycle through the three LEDs, lighting them one at a time.*/

    //SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);

    /*Before calling any peripheral specific driverLib function, enable the clock
    for that peripheral. If not enabled, it will result in a Fault ISR (address fault).
    This is a common mistake for new Stellaris users.*/
```

```
BLINK_LED
CORTEX_M4_0: GEL Output:
Memory Map Initialization Complete
```

Fig. 1.15. CCS debugging screen.

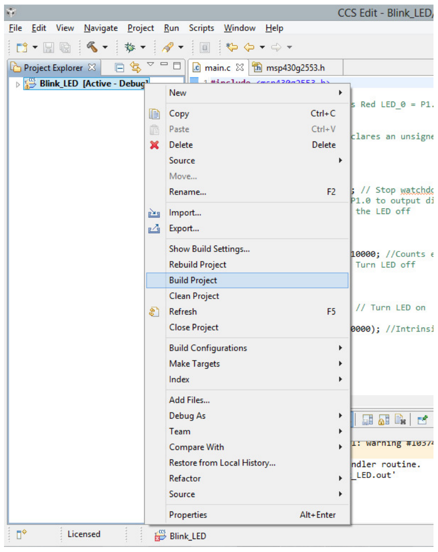


Fig. 1.16. CCS build project manually.

The main debugging tools are located along the row adjacent to the tab titled “Debug”. Fig. 1.17 shows the button locations and functions. The main tools used almost every debugging session are resume, stop, step into, and restart. The step function allows the code to execute one line at a time. Resume starts the code at the beginning of main runs through the code in real-time. Restart stops the code execution, points at the first line in main, and waits for debug termination, stepping, or resume. Stop or termination ends the debug session. To make a code change while in debug, pause or suspend the session, make the correction, save main.c, and press resume or step. CCS will rebuild the program, flash the MCU, and then execute the code. If the debug tab ever accidentally closes, the debug controls will close also. The debug pane is restored by clicking “View” and “Debug” on the menu bar. If the ULP advisor open’s a message, click proceed.
Note: It is possible the debugger cannot locate a file for execution. The file is located in the StellarisWare directory. **"C:\StellarisWare"**

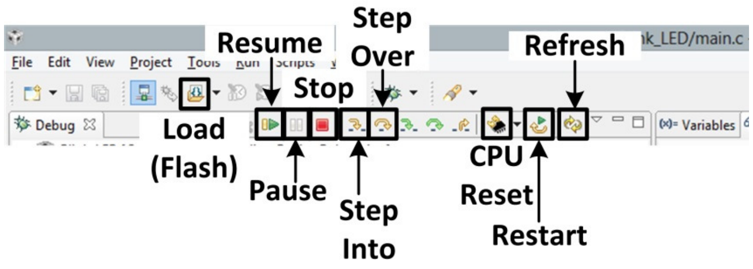


Fig. 1.17. CCS debug tools.

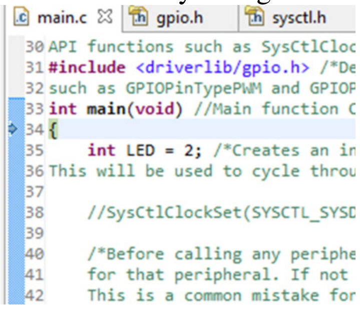
1. A small blue arrow left of the opening brace of main() in the middle window, see Fig. 1.18, points at the beginning of main(). This blue arrow indicates where the Program Counter (PC) is pointing to.

2. Use the “Step Into” button (or F5) to begin executing the code on the MCU one line a time.

3. Use the restart button and then click resume. The red LED will start blinking once every second.

4. Press the terminate button (or Ctrl+F2) to end the session.

5. Once the debug session has ended, notice the MCU continues executing the code flashed into memory. Congratulations, you have just successfully programmed and flashed a microcontroller!



```
API functions such as SysCtlCloc
#include <driverlib/gpio.h> /*De
such as GPIOPinTypePWM and GPIOF
int main(void) //Main function C
{
    int LED = 2; /*Creates an in
This will be used to cycle throu

    //SysCtlClockSet(SYSCTL_SYSD

    /*Before calling any periphe
    for that peripheral. If not
    This is a common mistake for
```

Fig. 1.18. CCS debug program counter arrow.

Advanced Debugging

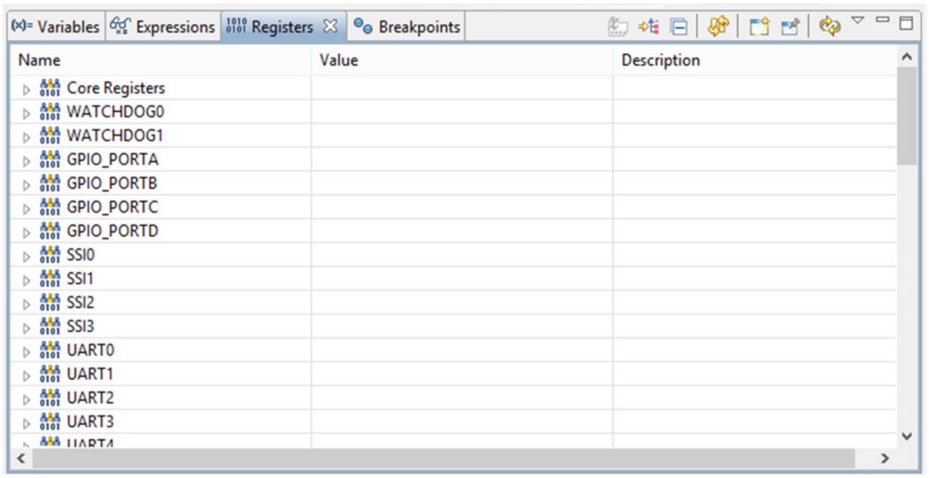
This section covers checking breakpoints, registers, and expressions. Restart the debug session.

Breakpoints

A breakpoint temporarily suspends the program execution until told to resume or step into the next line. Fig. 1.18 shows a blue bar running vertically starting at main and continues to the program's end. This blue bar shows where the PC arrow is positioned in the program and also sets breakpoints. Double click on the blue bar next to the line **`SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);`** and a blue ball with cross-hairs behind it will appear. In addition, the tab labeled "Breakpoints" appears in the upper right pain. Click on the tab and the newly created breakpoint is listed. The breakpoint maybe disabled by unchecking the box next to under the "Breakpoints" tab. Multiple breakpoints maybe used and disabled during debug.

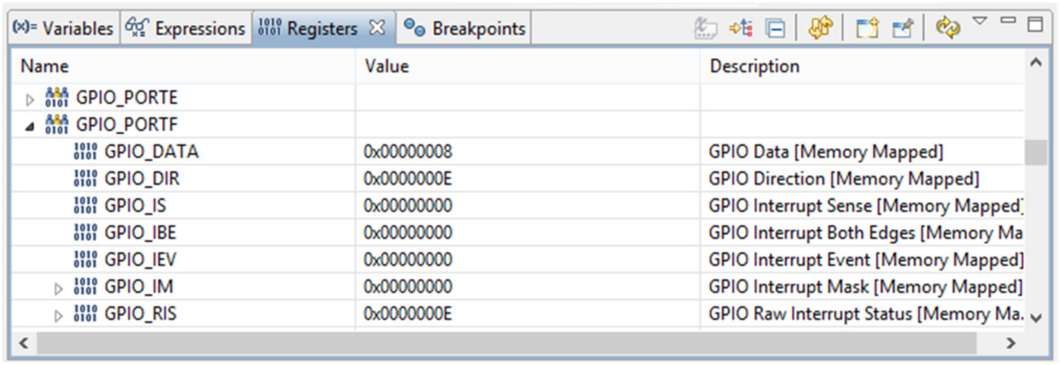
Registers

Debugging code or verifying data is read and solved correctly requires visual access to the registers on the Stellaris. The Stellaris registers tab is located in the CCS debug upper right pane as shown in Fig. 1.19. All the registers in the Stellaris are available for viewing. Fig. 1.20 shows the GPIO_PORTF register expanded and examining GPIO_DIR register values. GPIO_PORTF register communicates with the MCU which port and I/O pin(s) will be used and will the pin contain an input or output signal. The highlighted code **`GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, PinData);`** has been executed and the command sets the GPIO_DATA register and highlights the changed value in yellow. Register changes may be viewed in real-time or stepping one line at time.



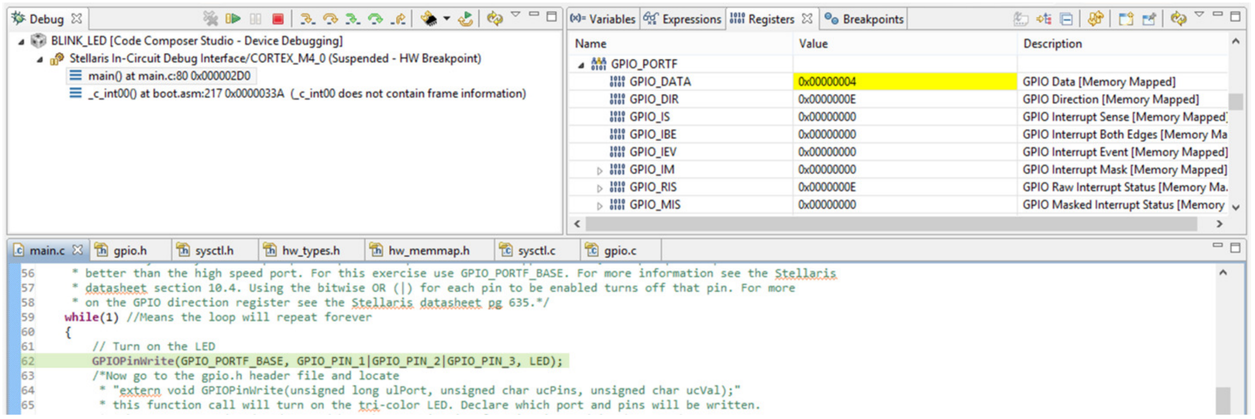
Name	Value	Description
Core Registers		
WATCHDOG0		
WATCHDOG1		
GPIO_PORTA		
GPIO_PORTB		
GPIO_PORTC		
GPIO_PORTD		
SSI0		
SSI1		
SSI2		
SSI3		
UART0		
UART1		
UART2		
UART3		

Fig. 1.19. CCS debug Stellaris register view.



Name	Value	Description
GPIO_PORTE		
GPIO_PORTF		
GPIO_DATA	0x00000008	GPIO Data [Memory Mapped]
GPIO_DIR	0x0000000E	GPIO Direction [Memory Mapped]
GPIO_IS	0x00000000	GPIO Interrupt Sense [Memory Mapped]
GPIO_IBE	0x00000000	GPIO Interrupt Both Edges [Memory Ma
GPIO_IEV	0x00000000	GPIO Interrupt Event [Memory Mapped]
GPIO_IM	0x00000000	GPIO Interrupt Mask [Memory Mapped]
GPIO_RIS	0x0000000E	GPIO Raw Interrupt Status [Memory Ma.

Fig. 1.20. GPIO_PORTF register expanded showing values for data and direction.



```
BLINK_LED [Code Composer Studio - Device Debugging]
  Stellaris In-Circuit Debug Interface/CORTEX_M4_0 (Suspended - HW Breakpoint)
    main() at main.c:80 0x000002D0
    _c_int00() at boot.asm:217 0x0000033A  (_c_int00 does not contain frame information)
```

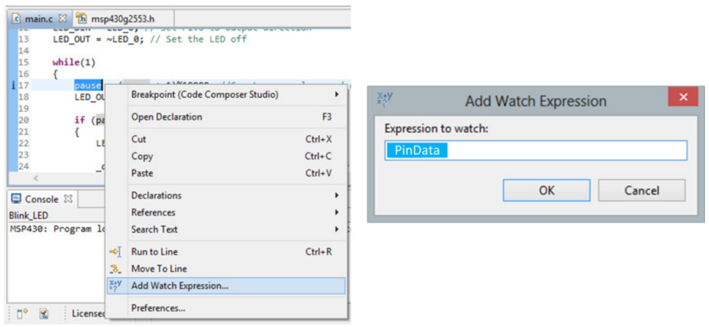
Name	Value	Description
GPIO_PORTF		
GPIO_DATA	0x00000004	GPIO Data [Memory Mapped]
GPIO_DIR	0x0000000E	GPIO Direction [Memory Mapped]
GPIO_IS	0x00000000	GPIO Interrupt Sense [Memory Mapped]
GPIO_IBE	0x00000000	GPIO Interrupt Both Edges [Memory Ma
GPIO_IEV	0x00000000	GPIO Interrupt Event [Memory Mapped]
GPIO_IM	0x00000000	GPIO Interrupt Mask [Memory Mapped]
GPIO_RIS	0x0000000E	GPIO Raw Interrupt Status [Memory Ma.
GPIO_MIS	0x00000000	GPIO Masked Interrupt Status [Memory

```
     * better than the high speed port. For this exercise use GPIO_PORTF_BASE. For more information see the Stellaris
     * datasheet section 10.4. Using the bitwise OR (|) for each pin to be enabled turns off that pin. For more
     * on the GPIO direction register see the Stellaris datasheet pg 635.*/
    while(1) //Means the loop will repeat forever
    {
        // Turn on the LED
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, LED);
        /*Now go to the gpio.h header file and locate
         * "extern void GPIOPinWrite(unsigned long ulPort, unsigned char ucPins, unsigned char ucVal);"
         * this function call will turn on the tri-color LED. Declare which port and pins will be written.
```

Fig. 1.21. GPIO_DATA register expanded showing changed values for data.

Expressions

Code statements such as loops, if statements, and math processing need verification to confirm the typed instructions perform as initially envisioned. These commands have indices, counters, and mathematical expressions containing integers. To examine an integer expression run the BLINK_LED program in CCS debug. Highlight the integer “PinData” to be tracked, right click, and select “Add Watch Expression” as shown in Fig. 1.22 and type in a name for the tracked expression and click “Ok”. Fig. 1.23 shows the expression “PinData” is visible under the expressions tab located in the upper right pane. Step through the program and notice the value change for "PinData" as the program moves past the while loop counter as shown in Fig. 1.24.

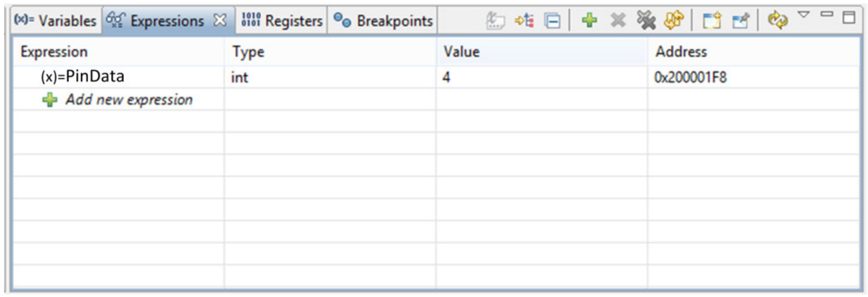


```
    LED_OUT = ~LED_0; // Set the LED off

    while(1)
    {
        pause
        LED_OU

        if (pa
        {
            LE

            _c
```

```
Console
Blink_LED
MSP430: Program lo
```

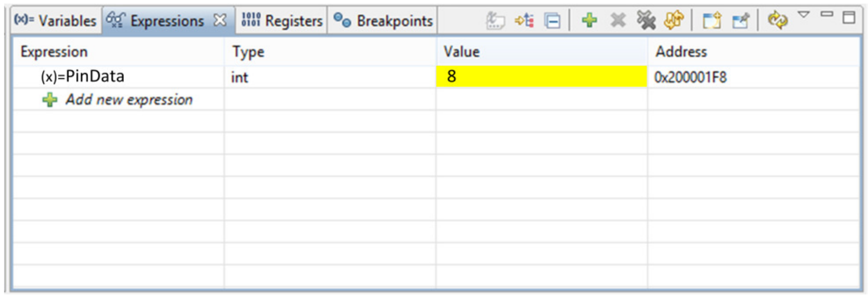
Expression to watch: PinData

Fig. 1.22. CCS debug adding expression to watch.



Expression	Type	Value	Address
(x)=PinData	int	4	0x200001F8
Add new expression			

Fig. 1.23. PinData expression added.



Expression	Type	Value	Address
(x)=PinData	int	8	0x200001F8
Add new expression			

Fig. 1.24. Stepping through the program shows the value pause change.

LM Flash Programmer

LM Flash Programmer is a standalone programming GUI that allows programing the flash memory of a Stellaris device through multiple ports. Creating the files required for the LM Flash is a separate build step in Code Composer that discussed next.

In the CCS Project Explorer, right-click on your project and select "Properties" and then on the left, click "Build" and then click the "Steps" tab. Paste the following commands into the "Post-build steps" "Command" box:

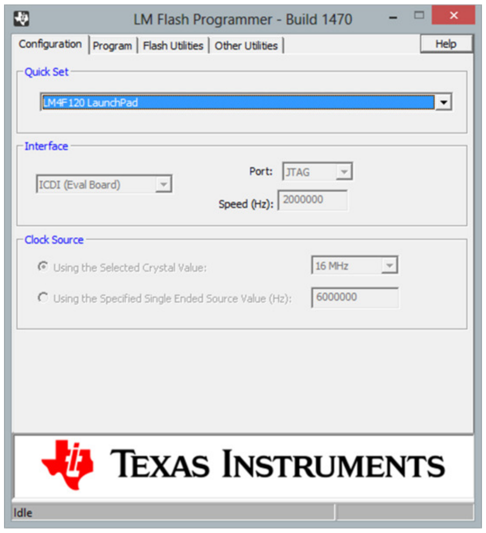
"${CCS_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin"
"${BuildArtifactFileName}"
"${BuildArtifactFileBaseName}.bin"
"${CG_TOOL_ROOT}/bin/armofd"
"${CG_TOOL_ROOT}/bin/armhex"
"${CCS_INSTALL_ROOT}/utils/tiobj2bin/mkhex4bin"

DO NOT PASTE THE COMMANDS UNTIL THE FOLLOWING STATEMENT IS READ!
Each command is enclosed by quotation marks and there is a space between each one. These steps will run after the project builds and the bin file will be in the debug folder. Access this file in the CCS Project Explorer and expanding the Debug folder.

Note: Exit from CCS debug or there will be a conflict between CCS and LF Flash for COM port control.

There is a shortcut to the LM Flash Programmer on the desktop to open the program. If not, look under the start menu, all programs, Texas Instruments, Stellaris, and LM Flash Programmer and click on LM Flash Programmer.

Click the Configuration tab. Select the LM4F120 LaunchPad from the Quick Set pull-down menu under the Configuration tab as shown in Fig. 1.25. Click on the Program tab. Then paste the following link:
C:\StellarisWare\boards\ek-lm4f120XL\qs-rgb\ccs\Debug\qs-rgb.bin
This is the original application programmed into the LM4F120XL flash memory during board assembly.



Quick Set: LM4F120 LaunchPad

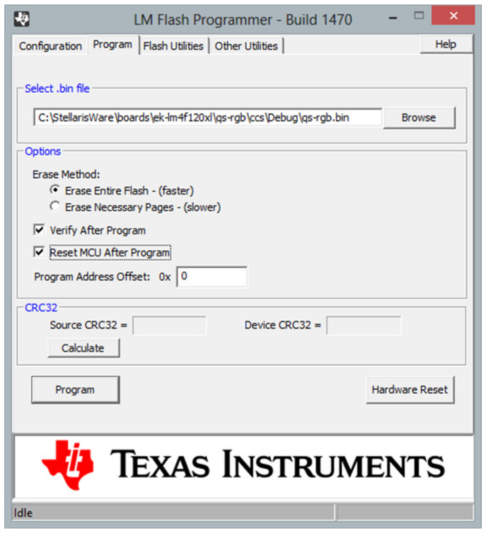
Interface: ICDI (Eval Board)
Port: JTAG
Speed (Hz): 2000000

Clock Source
Using the Selected Crystal Value: 16 MHz
Using the Specified Single Ended Source Value (Hz): 6000000

Idle

Fig. 1.25. LM Flash Programmer configuration menu.

Then select the checkboxes shown in Fig. 1.26 and finally click the "Program" button. The programming and verification status is located in the status bar at the window bottom. Once the flash programming has completed, the tri-color LED trial program will be running on the Launchpad. Close the LM Flash Programmer when finished.



Select .bin file: C:\StellarisWare\boards\ek-lm4f120xl\qs-rgb\ccs\Debug\qs-rgb.bin

Options

Erase Method:
- [✓ Erase Entire Flash - (faster)]
- Erase Necessary Pages - (slower)

[✓ Verify After Program]

[✓ Reset MCU After Program]

Program Address Offset: 0x 0

CRC32

Source CRC32 = Device CRC32 =

Fig. 1.26. LM Flash programmer setup.

Second Project

Introduction to StellarisWare®, Initialization and GPIO

Complete the clocks and GPIO lab located in chapter three of the Stellaris Workshop Manual.
Stellaris Workshop Manual http://software-dl.ti.com/trainingTTO/trainingTTO_public_sw/GSW-Stellaris-LaunchPad/StellarisLaunchPadWorkbook.pdf

Third Project

Interrupts and the Timers

Complete the interrupts and timers lab located in chapter four of the Stellaris Workshop Manual.
Stellaris Workshop Manual http://software-dl.ti.com/trainingTTO/trainingTTO_public_sw/GSW-Stellaris-LaunchPad/StellarisLaunchPadWorkbook.pdf

Final Project

After acquiring knowledge of clocks, interrupts, and timers from the completed exercises above; now it's time for application and the gathering of wisdom. **You must combine and apply clocks and timers and interrupts** to blink the tri-color LED and integrate a button interrupt to turn off the tri-color LED and use a second button to change the flashing tri-color LED rate. Demonstrate using functional C programming and proper programming technique, debugging, low power programming, and disable unused ports and interrupts. Answer all questions and supply your own comments for the code.